UNITED STATES PATENT APPLICATION FOR:

METHOD AND APPARATUS FOR PROCESSING NETWORK PACKETS

INVENTORS:

PATRICK DENIS LINCOLN
STEVEN EKER

ATTORNEY DOCKET NUMBER: SRI/4804-2

CERTIFICATION OF MAILING UNDER 37 C.F.R. 1.10

I hereby certify that this New Application and the documents referred to as enclosed therein are being deposited with the United States Postal Service on March 15, 2004, in an envelope marked as "Express Mail United States Postal Service", Mailing Label No. EV 367981113 US, addressed to: Commissioner for Patents, Mail Stop PATENT APPLICATION, P.O. Box 1450, Alexandria, VA 22313-1450

Linda DeNardi
Signature

Linda DeNardi
Name

March 15, 2004
Date of signature

MOSER, PATTERSON & SHERIDAN LLP
595 Shrewsbury Ave.
Shrewsbury, New Jersey 07702
(732) 530-9404

METHOD AND APPARATUS FOR PROCESSING NETWORK PACKETS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims benefit of United States provisional patent application serial number 60/454,935, filed March 13, 2003, which is herein incorporated by reference.

BACKGROUND OF THE INVENTION

Field of the Invention

[0002] The present invention relates to computer networks. More specifically, the present invention relates to a method and apparatus for monitoring and detecting strings of interest to effect intrusion detection, packet filtering, load balancing, routing, and other network-related operations as disclosed below.

Description of the Related Art

[0003] Network packets, e.g., those involved in TCP/IP or UDP communications, are routinely split into network datagrams for transmission on a network medium. Standard methods for performing intrusion detection, packet filtering, load balancing, routing, and other network-related operations involve the reassembly of network datagrams into network packets in order to make decisions based on the contents of the packet. As the splitting of packets can occur when each higher-layer in the network stack passes its data to a lower-layer, such reconstruction may be applied at any of such reconstructive boundaries, for example: IP packets being split into network datagrams, TCP packets being split into IP packets, HTTP packets being split into TCP packets, and others as will be clear to those skilled in the art.

[0004] Many of such decisions involve searching for particular strings that occur in the monitored network packets. To perform network monitoring, it is often necessary to completely reassemble higher layer (e.g., TCP/IP) packets from the monitored network datagrams in order to effectively match the strings of

interest. This is necessary in order to counteract malicious avoidance of the monitoring system by splitting the strings between packets, and the out of order arrival of packets belonging to a particular network activity. However, reassembly of packets requires significant processing and storage, such that traditional methods are unable to scale cost-effectively to modern high-speed networks. Further, by malicious act, the monitoring system may be forced to retain large amounts of state, resulting in a denial of service attack.

[0005] Thus, there is a need for a method and apparatus for monitoring and detecting strings of interest to effect intrusion detection, packet filtering, load balancing, routing, and other network-related operations without the need to completely reassemble higher layer packets.

SUMMARY OF THE INVENTION

[0006] In one embodiment, the present invention discloses a method and apparatus for monitoring and detecting strings of interest to effect intrusion detection, packet filtering, load balancing, routing, and other network-related operations without the need to completely reassemble higher layer packets. Specifically, the present invention described herein relates to methods for monitoring, repairing, and responding to network activity based on a string-matching method that finds all occurrences of words from a given finite language in a sequence of datagrams, where blocks of data may arrive out-of-order. It has utility in performing content analysis on TCP/IP streams without either TCP stream reassembly or IP datagram reassembly. It has further utility in performing error-tolerant string matching: for fixed-vocabulary streams, the present invention can repair packets, or completely patch in missing packets from a sequence of datagrams, thereby preventing the recipient of a communication from requesting retransmission of missing parts of the sequence.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] So that the manner in which the above recited features of the present invention can be understood in detail, a more particular description of the

invention, briefly summarized above, may be had by reference to embodiments, some of which are illustrated in the appended drawings. It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the invention may admit to other equally effective embodiments.

[0008] Figure 1 illustrates a block diagram of a network of the present invention;

[0009] Figure 2 illustrates a flowchart of method for substring matching; and

[0010] Figure 3 illustrates a flowchart of method for monitoring and detecting fragments of strings of interest and/or strings of interest in the present invention.

[0011] To facilitate understanding, identical reference numerals have been used, wherever possible, to designate identical elements that are common to the figures.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0012] The present invention relates to computer networks. More specifically, the present invention relates to a method and apparatus for monitoring and detecting strings or substrings of interest to effect intrusion detection, packet filtering, load balancing, routing, and other network-related operations as disclosed below.

[0013] Figure 1 illustrates a block diagram of an exemplary network 100 of the present invention. Figure 1 illustrates a plurality of computing devices 110-130 (e.g., general purpose computers, work stations, servers, storage devices, PDAs, and other computing devices in general) that are organized into a network.

[0014] Computing device 120 is illustrated as having a central processing unit (CPU) or a controller 122, a memory 124, a plurality of input/output (I/O) devices 126 and a string detection module 128. The input/output (I/O) devices 126 may comprise storage devices (e.g., a tape drive, a floppy drive, a hard disk drive or a compact disk drive), a receiver, a transmitter, a speaker, a display, an image capturing sensor, a clock, an output port, a user input device (such as a keyboard, a keypad, a mouse, and the like, or a microphone for capturing speech commands), and the like. It should be understood that the string detection module 128 can be implemented as one or more physical devices that

are coupled to the CPU 122 through a communication channel. Alternatively, the string detection module 128 can be represented by one or more software applications (or even a combination of software and hardware, e.g., using application specific integrated circuits (ASIC)), where the software is loaded from a storage medium, (e.g., a magnetic or optical drive or diskette) and operated by the CPU 122 in the memory of the computing device. As such, the string detection module 128 (including associated data structures) of the present invention can be stored on a computer readable medium, e.g., RAM memory, magnetic or optical drive or diskette and the like. Additionally, although computing devices 110a-n and 130 are not shown with a processor, a memory, a plurality of input/output (I/O) devices and a string detection module, it should be noted that these computing devices can deploy these components as well.

[0015] In the illustrative embodiment of FIG. 1, computing device 120 is deposed between computing device 130 and the plurality of computing devices 110a-n. Thus, as disclosed in greater detail below, computing device 120 can be deployed as an intrusion detection system, a firewall, a router, a switch, a load balancer, an anti-virus filter, an anti-spam filter, a document control system, a web content filter, one or more virtual private network devices, one or more SAN security devices and the like. A brief description is provided below as to how the present string matching can be advantageously employed in each of these embodiments. Thus, the architecture as shown in FIG. 1 is only illustrative of the present invention and should not be interpreted as limiting the present invention to this particular configuration.

[0016] In intrusion detection, the strings of interest correspond to signatures of malicious activity, intended to perform such actions as buffer overflows in applications, to confuse the network stack; and so on. Malicious actors often employ deliberate tricks of splitting the malicious string over several network datagrams, or forcing the datagrams to arrive out of order, in an attempt to avoid detection by such a monitor.

[0017] Actions and responses in intrusion detection systems could involve reporting the matched strings to a monitoring console, or to a correlation

engine, for example as described in US Patent 6,484,203 to Porras et al, and assigned to the assignee of the present invention and herein incorporated by reference. Further responses include terminating the connection on matching a string, for example by transmitting a TCP reset (RST) packet or an ICMP message.

[0018] In a firewall, the goal is to prevent certain network flows from entering a protected network. Traditional methods either involve the reconstruction of complete application-layer packets, and performing allow/deny decisions based on analysis thereof, or reassembly to just the transport layer (e.g., TCP or UDP) and performing allow/deny decisions based on port numbers and other network address information, according to a predefined security policy.

[0019] Clearly therefore, firewalls suffer from the same overheads of packet reconstruction as previously described. According to the present invention, the present invention can make allow/deny decisions based on individual network datagrams, by monitoring for strings defined in accordance with the predefined security policy. The allow/deny decisions are then based on a per received datagram basis. Although parts of a higher-level network packet will be allowed to pass the firewall, the present invention advantageously allows the firewall to block merely the last datagram that completes a match for one of the predetermined strings, and optionally perform a further response action, such as resetting a connection, raising an alarm, or invoking countermeasures. As the destination of the communication requires all of the network datagrams in order to completely reconstruct the (malicious) network packet, by blocking one, the present invention can prevent such reconstruction and thereby prevent any possible damage to the destination computer.

[0020] Routers and switches monitor the contents of network packet "header" information in order to make decisions regarding where to retransmit the received network packet. Some switches operate at the datagram level, but these types of switches perform merely low-level operations, usually on Ethernet frames, in order to transmit the frame to the intended destination network link.

[0021] More capable and intelligent routers and switches involve making decisions at higher levels of the network stack, and reconstruct the higher level packets from the underlying network datagrams to perform this operation. The present invention can be advantageously employed in routers and switches to monitor for strings according to a switching or routing policy, and action taken in the form of retransmitting the datagram on the appropriate outbound network link or links.

[0022] In cases where one of the strings being monitored for, that forms the basis of a routing or switching decision, is found only in a first datagram of a network packet, it is possible for the later datagrams to arrive before the first datagram of the network packet. One could buffer these later datagrams in memory until the first datagram (the decision point) arrives. According to the present invention, and the underlying concepts of making per-datagram decisions, one could route or switch the early arriving datagrams on all network links.

[0023] Load balancing devices are similar to routers, with the additional constraint in the routing decision to balance the load across multiple devices that perform similar functions. For example, in Web-server load balancing systems, the decision may in part be based upon a URL contained in a request for a web page. Reconstructing application layer packets, such as HTTP protocol packets, to perform this decision making can be avoided by employing the present invention to monitor for strings, such as fragments or prefixes of URLs in the case of a Web-server load balancing system.

[0024] Many modern virus detection programs utilize signatures to detect the presence of a particular virus in a file. String matching according to the present invention can be used to perform anti-virus filtering without the requirement to reassemble the whole file (as represented in the network stream), by monitoring for strings commonly occurring in virus infected files. The present invention could be utilized to monitor, report and eliminate viruses at any point in the network, for example at a firewall, gateway, router, switch, or on a server or desktop computer.

[0025] The removal of unsolicited commercial email, commonly referred to as spam, is another potential utility of the present invention. Many spam messages have distinct strings in their content or message headers that enable them to be differentiated from legitimate electronic mail. The present invention could be utilized to monitor, report and eliminate certain spam messages at any point in the network, for example at a firewall, gateway, router, switch, or on a server or desktop computer.

[0026] Document control systems are, in one aspect, used to monitor the infiltration and exfiltration of information. By using the present invention to monitor network traffic for strings indicating confidential handling requirements for a document, topic material, or other marking of interest, the malicious or accidental transmittal of such documents outside an organizational unit may be prevented.

[0027] Web content filters are devices that remove potentially harmful content requested by users (with or without their explicit consent) during a browser session. Such content includes, but is not limited to, Java applets, JavaScript programs, and Visual Basic scripts. Further filtering may be performed based on objectionable content according to some policy, for example prohibiting the viewing of pornographic material, or visiting particular Web sites.

[0028] The present invention can be utilized in monitoring virtual private networks, and other devices such as SSL accelerators that involve the use of cryptography to secure communications between two or more parties. The packet-monitoring phase of the method can be extended to include the decryption of the monitored packet, if the key for decrypting is available to the monitor.

[0029] SANs (Storage Area Networks) tend to use fibre channel based networking, which is faster than many traditional forms of networks and hence traditional security monitoring and response systems are unable to perform effectively.

[0030] There are several points of security concern in a standard SAN: Admin-to-SAN management, Load Balancing, Server-to-switch, Switch-to-switch, Switch-to-RAID and RAID to backing store. Each of these concerns is addressed separately below.

[0031] Admin-to-SAN management may be setup in various ways, including using IP-sec tunneling or other specific configuration. The present invention may be used to detect, report and block administrative commands coming through unauthorized channels or from unauthorized locations by monitoring for the strings corresponding to the administrative commands.

[0032] The SAN load balancing function distributes storage and retrieval requests across multiple servers in a SAN, by interpreting the contents of the storage or retrieval commands contained in the network packets. The load-balancing component is mission-critical, and can be a bottleneck due to having to reassemble application-layer traffic to perform application-layer routing decisions. Also, the load-balancing component may be a victim of a denial of service attack by malicious agents or accidental misconfiguration. The present invention can enable most of the application-layer decisions required here without reassembly by monitoring for the strings representative of the storage and retrieval commands, thus significantly reducing the overhead in the load-balancing component.

[0033] In the server to switch interface in a SAN, server devices ports are bound to a set of fibrechannel ports using access control lists (ACLs). The configuration and reconfiguration of these ports and ACLs is critical to reliable operation, and in fact to any operation at all. The present invention can advantageously monitor the strings corresponding to the configuration and ACL commands for this interface.

[0034] The switch-to-switch interface enables reconfiguration of routing and control, which must be monitored. According to the present invention, one can provide filtering of reconfiguration commands based on matching the corresponding strings, or can provide audit trails and management alerts relating operations monitored. The advantageous scalability of the present

invention is critical for monitoring this stage of a SAN, as the switches are incapable of buffering the 2Gbps or greater traffic loads due to space and time constraints, and thus traditional approaches are impractical.

[0035] The switch-to-RAID interface connects the switches to the disk arrays (RAID, for Redundant Array of Inexpensive Disks). This interface is often unprotected in SAN deployments, which may be acceptable in secured locations, but in shared facilities such as collocation and data centers, this interface will require monitoring, reporting and response. The present invention can monitor for strings representing the storage, retrieval and reconfiguration requests.

[0036] The backing store to RAID interfaces enables long-term archival storage and backup mechanisms in SAN deployments. The present invention can enable security of this interface by monitoring the backup process for predefined markers in the backup information flow.

[0037] The present string matching method is now disclosed. A description of the string matching method is generically described first and then is further described within the context of a flowchart of Figure 2.

[0038] Let u and v be strings. The present invention denotes the length of u by $|u|$ and the concatenation of u and v by u.v.

[0039] The present invention assumes a finite language L of bad words with lengths less than or equal to k. It is also assumed the existence of a finite state automaton A to recognize members of L with arbitrary leading and trailing context. The present invention denotes the initial state of A as initial_A and the result of running A on a string d, starting at state c as $\text{run}_A(c, d) = \langle c', m \rangle$ where c' is the new state reached in A and m is the set of matches found.

[0040] In the present model, the present invention considers a TCP segment t to be a triple $\langle a, q, d \rangle$ where a identifies a unique TCP stream, q is the sequence number and d is the data, comprising a string of bytes. The segment t is said to be short if and only if $|d| < k-1$.

[0041] For each active TCP stream, the present invention maintains a span set where each span is a 4-tuple s=<b, l, p, c>. A span corresponds to a sequence of one or more contiguous TCP segments that have been processed and sent on their way. Here b is the sequence number of the first segment in the sequence, l is the total length in bytes of the data in those segments, p is a prefix comprising a string of up to k-1 bytes from the beginning of the data and c is a state from A. The span s is said to be short if and only if $l < k-1$.

[0042] It is convenient to define a family of three functions, *join*, that take respectively, a span and string, a string and a span, or two spans and return a new span and a set of matches.

[0043] Definition 1:

$\mathrm{join}(\langle b_1, l_1, p_1, c_1\rangle, d) = \langle\langle b_3, l_3, p_3, c_3\rangle, m\rangle$ where

$\langle c_3, m \rangle = \mathrm{run}_A(c_1, d)$

$b_3 = b_1$

$l_3 = l_1 + |d|$

p_3 is the $\min(k-1, |p_1 . d|)$ byte prefix of $p_1 . d$

[0044] Definition 2:

$\mathrm{join}(d, \langle b_2, l_2, p_2, c_2 \rangle) = \langle\langle b_3, l_3, p_3, c_3 \rangle, m\rangle$ where

$\langle c, m \rangle = \mathrm{run}_A(\mathrm{initial}_A, d . p2)$

$b_3 = b_2 - |d|$

$l_3 = |d| + l_2$

p_3 is the $\min(k-1, |d . p_2|)$ byte prefix of $d . p_2$

$c_3 = c$ (if $l_2 < k-1$) or c_2 (otherwise)

[0045] Definition 3:

$\mathrm{join}(\langle b_1, l_1, p_1, c_1\rangle, \langle b_2, l_2, p_2, c_2\rangle) = \langle\langle b_3, l_3, p_3, c_3\rangle, m\rangle$ where

$< c, m > = run_A(c_1, p_2)$

$b_3 = b_1$

$l_3 = l_1 + l_2$

p_3 is the min(k-1, $|p_1 . p_2|$) byte prefix of $p_1 . p_2$

$c_3 = c$ (if $l_2 < k-1$) or c_2 (otherwise)

[0046] When a segment < a, q, d > arrives, the present invention looks in the associated span set to see if the spans $s_p = < b_p, l_p, p_p, c_p >$ immediately preceding the segment, and $s_s = < b_s, l_s, p_s, c_s>$ immediately succeeding the segment exist. The present invention considers the following four cases:

[0047] Case 1: Neither s_p nor s_s exist. Let $< c, m > = run_A(initial_A, d)$. The present invention reports matches m and inserts a new span < q, |d|, p, c > into the span set, where p is the min(k-1, |d|) byte prefix of d.

[0048] Case 2: s_p exists, s_s does not. This is the usual case, when segments arrive in order. Let < s, m > = join(sp, d). The present invention reports matches m and replaces s_p with s.

[0049] Case 3: s_s exists, s_p does not. Let <s, m> = join(d, s_s). The present invention reports matches m and replaces s_s with s.

[0050] Case 4: Both s_p and s_s exist. Let <s, m> = join(s_p, d). Let <s', m' > = join(s, s_s). The present invention reports matches $m \cup m'$ and replaces both s_p and s_s with s'.

[0051] Figure 2 illustrates a flowchart of the string matching method 200 as outlined above. Namely, method 200 can be deployed by computing device 120 to effect various string detection implementations as discussed above. Method 200 starts in step 205 proceeds to step 210.

[0052] In step 210, method 200 reads a substring index pair <d, i>, where d is the substring and i is broadly defined as an index. It should be noted that

although the present invention is disclosed within the context of TCP/IP, the present invention is not limited to a particular protocol. Namely, the present invention can be implemented in conjunction with other communication protocols, e.g., UDP and so on. Thus, a substring should be broadly interpreted as representing a string, a packet, a segment and so on. Similarly, index i should be broadly interpreted as representing a position within a sequence, e.g., a sequence number such as q in the context of TCP/IP.

[0053] In step 215, method 200 queries whether a preceding span s_p is in a span set. If the query is negatively answered, then method 200 proceeds to step 220. If the query is positively answered, then method 200 proceeds to step 230.

[0054] In step 220, method 200 queries whether a succeeding span s_s is in a span set. If the query is negatively answered, then method 200 proceeds to step 221. If the query is positively answered, then method 200 proceeds to step 224.

[0055] In step 221, method 200 having concluded that there are no preceding span s_p and no succeeding span s_s, applies the automaton A to output and record any matches m. Method 200 then inserts a new span <i, |d|, p, c> into the span set. It should be noted that steps 221-223 reflects case 1 as discussed above. Method 200 then returns to step 210 for the next substring index pair.

[0056] In step 224, method 200 performs a join operation where the substring d is joined with the succeeding span s_u. The string matching method of the automaton A is applied and method 200 outputs and records any matches m in step 225. Finally, method 200 replaces the succeeding span s_u with the joined span s. It should be noted that steps 224-226 reflects case 3 as discussed above. Method 200 then returns to step 210 for the next substring index pair.

[0057] In step 231, method 200 performs a join operation where the preceding span s_p is joined with the substring d. The string matching method of the automaton A is applied and method 200 outputs and records any matches m in

step 232. Finally, method 200 replaces the preceding span s_p with the joined span s. It should be noted that steps 231-233 reflects case 2 as discussed above. Method 200 then returns to step 210 for the next substring index pair.

[0058] In step 234, method 200 performs a join operation where the preceding span s_p is joined with the substring d. In step 235, method 200 performs a join operation where the join span s from step 234 is joined with the succeeding span s_s. The string matching method of the automaton A is applied and method 200 outputs and records any matches $m \cup m'$ in step 236. Finally, method 200 replaces the preceding span s_p and the succeeding span s_s with the joined span s'. It should be noted that steps 234-237 reflects case 4 as discussed above. Method 200 then returns to step 210 for the next substring index pair.

[0059] It should be noted that the present substring method broadly describes an approach where substrings are rapidly processed and forwarded without having to store and reassemble all the substrings. Only various parameters of the strings are stored such as the state of the automaton, an index, e.g., a sequence number, a length of the substring and a prefix. These stored parameters will allow the present invention to quickly detect a substring of interest.

[0060] It should be noted that the present method assumes that whole TCP segments are received as input. However TCP segments travel the Internet in IP datagrams that can be fragmented and refragmented as dictated by the MTU of the physical networks over which they travel and the fragments themselves may arrive out-of-order.

[0061] If it is desired to inspect the TCP header before deciding whether to do string matching on the data in a TCP segment then it may be necessary to perform IP datagram reassembly. However, if it is desired to perform string matching on the contents of all TCP streams, then the present invention can be adapted to work with out-of-order IP datagram fragments.

[0062] In one embodiment, the present invention makes the simplifying assumption that the IP fragment with offset 0 contains at least 60 bytes of data

following the IP header. This ensures that the TCP header fits completely in this fragment (since TCP headers have a maximum length of 60 bytes) and can be recognized when it arrives.

[0063] The source IP address, destination IP address and IP identification fields in the IP fragment header uniquely define which IP datagram each fragment belongs to. For each incomplete datagram, the present invention uses the basic method above at the IP level to maintain a set of spans where each span represents a contiguous sequence if IP fragments. The fragment with offset 0 is handled specially, where the invention extracts the TCP header and does not treat it as data. Any matches found must be stored until the whole datagram has been seen. Once all the fragments for a given IP datagram have been processed, the present invention will be left with a TCP header, a single span and a (possibly empty) list of matches. From the TCP header, the present invention can calculate the position of each match relative to the TCP stream. Using the present method at the TCP level is now slightly more complicated since for each TCP segment, instead of a string of bytes d, there is a span $s_{new} = < b, l, p, c >$ where b and l are determined from the TCP header and p and c are from the remaining span at the IP level. The present invention looks in the span set associated with the TCP stream to see if the spans $s_p = < b_p, l_p, p_p, c_p >$ immediately preceding the segment, and $s_s=<b_s\ l_s,p_s,c_s>$ immediately succeeding the segment exist.

[0064] Consider the same 4 cases as above.

[0065] Neither s_p nor s_s exist. The present invention adds s_{new} to the span set.

[0066] s_p exists, s_s does not. Let $<s, m> = join(s_p, s_{new})$. The present invention reports matches m and replaces s_p with s.

[0067] s_s exists, s_p does not. Let $< s, m > = join(s_{new}, s_s)$. The present invention reports matches m and replaces s_s with s.

[0068] Both s_p and s_s exist. Let $< s, m > = join(s_p, s_{new})$ and $<s', m' > = join(s, s_s)$. The present invention reports matches $m \cup m'$ and replaces both s_p and s_s with the single span s'.

[0069] Note that if one wants to avoid bad words getting through, the present invention should not retransmit the last-to-arrive IP fragment until the TCP level analysis has been completed.

[0070] The case where an IP fragment with offset 0 contains fewer than 60 bytes of data will be rare and will often be an indication of an attack. The present invention detects such a situation when either a fragment with offset 0 arrives and it has fewer than 60 data bytes, or when a fragment with offset greater than 0 but less than 60 arrives. At this point the present invention switches to a slower mode of operation where it explicitly reassembles the first 60 bytes of data from the IP datagram in a buffer.

[0071] An IP fragment containing data that straddles the 60 byte boundary can be split in two - the data bytes inside the 60 byte boundary go into the buffer while the bytes outside the boundary are processed as usual. When all the fragments have been seen the present invention will have a single span together with a 60 byte buffer. The buffer can be inspected to determine where the TCP header ends and the remaining data bytes from the buffer can be processed as usual to get a final span at the IP level.

[0072] Where L is explicitly given, the Aho-Corasick algorithm (see A. Aho and M. Corasick, Efficient String Matching: An Aid to Bibliographic Search, Communications of the ACM 18(6):333-343, June 1975) or similar algorithm can be used to construct an O(m) state automaton where m is the sum of the lengths of the words in L. The running time of the automaton on a string of length n is O(n + j) when j is the number of matches found. A number of algorithms attempt to combine the Boyer-Moore technique (see R.S. Boyer and J S. Moore, A Fast String Searching Algorithm, Communications of the ACM 20(10):762-772, October 1977) with the Aho-Corasick algorithm to obtain a sublinear running time. However such algorithms are more complex and need to test symbols in a nonlinear order. Furthermore, their practical advantage

over the Aho-Corasick algorithm evaporates if just one member of L is short. An Aho-Corasick algorithm can be converted to a deterministic finite-state automaton (DFA) in O(m) time. This has the advantage of removing failure arcs that halves the worst-case number of transitions and simplifies the execution. However it has the disadvantage that each state will typically have a larger transition function now that there is a success arc for each symbol in the alphabet.

[0073] Where L is given by a regular expression it may be feasible to build a deterministic finite-state automaton for L by classical techniques.

[0074] By storing a span set as a balanced binary tree such as a red-black tree (see for example Chaper 14 of Thomas H. Cormen, Charles E. Leierson and Ronald L. Rivest, "Introduction to Algorithms", MIT Press and McGraw Hill, 1990), finding the spans preceding and succeeding a segment, and inserting and deleting spans can all be done in $O(\log n)$ time where n is the number of spans in the set. In practice however, the number spans in a given set is likely to be small so the present invention could choose some cutoff value, C, and regard TCP streams generating more than C spans as an attack on the IDS itself by a knowledgeable attacker. When the number of spans is small, storing the span set as a singly linked list allows for faster operation, especially on today's highly pipelined architectures, since the branches will be more predictable in linked list code than in balanced binary tree code.

[0075] An alternative approach to storing the automaton state c in each span set is to instead store a suffix z of length at most k-1. When c is required, for example in a join operation, the present invention can recover it by $\langle c,m \rangle = \mathrm{run}_A(\mathrm{initial}_A,z)$, and discard the (already reported) matches m. However, this approach requires more storage than the preferred embodiment already described (if k-1 is greater than the size of the pointer or integer required to store a state), requires more cases to code in the algorithm, and more time.

[0076] An Aho-Corasick automaton is essentially a trie with the addition of a failure arc from each state and a (possibly empty) list of matches for each state. The present invention defines the depth of a state as its distance from the root

in the underlying trie. In the average case, states of shallow depth tend to have many success arcs in the underlying trie when as states of large depth typically have 0 or 1.

[0077] Alternative techniques for representing states include:

1. Represent success arcs as a balanced binary tree of next states with byte value keys.

2. Represent success arcs as an array of 16-bit state numbers indexed by byte values.

3. Represent success arcs as an array of 32-bit state numbers indexed by byte values.

4. Represent success arcs as an array of state pointers indexed by byte values.

5. Use one of 2-4 for "big" states (with more than one success arc) and store a (value, state number/pointer) pair for "small" states (with 0 or 1 success arcs). In the 0 success arc case the value is an out-of-range value to force taking the failure arc without an explicit test for the 0 success arc case.

6. Compile the states to code fragments via C/C++.

[0078] Another possibility is to convert the Aho-Corasick automata into a DFA and remove failure arcs altogether. This approach simplifies the inner loop at execution time and removes a potentially unpredictable branch that can hurt performance on modern highly pipelined architectures. The downside is that each state now has a success arc for every byte value so technique 5 above can't be used and techniques 1 and 6 may be very expensive in both time and space.

[0079] One optimization that is often applied to DFAs, and that can be applied as an extension of the methods disclosed herein, is that of character class compression. (See Vern Paxon, *Flex – a scanner generator*, March 1995.) For

example characters that don't appear in any bad word are indistinguishable for the purpose of matching, and could all be represented by a single byte value.

[0080] In one embodiment, the present invention is adapted to address dropped packet attack by monitoring both sides of the TCP communication. Namely, if an attacker can control or predict dropped packets inside the firewall, the attacker may attempt to circumvent the protection by sending a stream with a bad word split across multiple packets. This is followed by sending one of the packets with benign filler. This packet is processed by the firewall, and the whole stream is found to be non-problematic. However, the "filler" packet is then forcibly dropped by the internal network before it is processed by the target machine. The target machine TCP stack then times out, and requests retransmission of the filler packet. The attacker then responds to this request with a packet that has (part of) harmful information. Since the present invention would have already forgotten the state of the other packets surrounding the filler, and the filler packet content, the newly sent packet cannot be properly checked for bad words.

[0081] To address this scenario, the present system can monitor both incoming and outgoing traffic on a given connection, e.g., a TCP connection. If there are any packets dropped past the present system, then one can force a reset of the connection, or simply drop "retransmitted" packets. In other words, this feature compares the TCP ACKs to see if there has been any dropped packets behind the present system, and if so, takes appropriate action such as sending a RST to the connection.

[0082] This approach should have no appreciable effect on normal traffic in preferred embodiments where the present invention is deployed at or near a firewall in front of a LAN, where packets are almost never dropped in the LAN. However, this approach will advantageously prevent the above-mentioned potential attack mode.

[0083] Figure 3 illustrates a flowchart of method 300 for monitoring and detecting fragments of strings of interest and/or strings of interest in the present invention. More specifically, method 300 can be deployed by computing device

120 to effect various string detection implementations as discussed above plus subsequent processing.

[0084] Method 300 starts in step 305 and proceeds to step 310. In step 310, a computing device receives a plurality of substrings, e.g., packets.

[0085] In step 320, method 300 applies fragment recognition to the received substrings as discussed in Figure 2 above.

[0086] In step 330, method 300 queries whether fragments of interest have been detected. If the query is affirmatively answered, then method 300 proceeds to step 340. If the query is negatively answered, then method 300 proceeds to step 380, where the received substrings are forwarded and processed normally within a particular implementation context.

[0087] In step 340, method 300 stores substrings that contain fragments of interest. These substrings are reassembled or partially assembled to a higher layer.

[0088] In step 350, method 300 again applies string recognition to the reassembled substrings as discussed in Figure 2.

[0089] In step 360, method 300 queries whether strings of interest have been detected. If the query is affirmatively answered, then method 300 proceeds to step 370. If the query is negatively answered, then method 300 proceeds to step 380, where the received packets are forwarded and processed normally within a particular implementation context.

[0090] In step 370, method 300 may report, reject, reroute, and/or run other higher order functions on the packets carrying strings of interest. Method 300 ends in step 385.

[0091] While the foregoing is directed to embodiments of the present invention, other and further embodiments of the invention may be devised without departing from the basic scope thereof, and the scope thereof is determined by the claims that follow.